Teemu Niemi

DirectX 11 -grafiikkamoottori ja BRDF-mallit

Opinnäytetyö
Kevät 2017
SeAMK Tekniikka
Tietotekniikan koulutusohjelma

SeAMK
SEINÄJOEN AMMATTIKORKEAKOULU
SEINÄJOKI UNIVERSITY OF APPLIED SCIENCES

SEINÄJOEN AMMATTIKORKEAKOULU

Opinnäytetyön tiivistelmä

Koulutusyksikkö: Seinäjoen Ammattikorkeakoulu

Tutkinto-ohjelma: Tietotekniikan koulutusohjelma

Suuntautumisvaihtoehto: Ohjelmistotekniikka

Tekijä: Teemu Niemi

Työn nimi: DirectX 11 -grafiikkamoottori ja BRDF-mallit

Ohjaaja: Petteri Mäkelä

Vuosi: 2017 Sivumäärä: 48

Opinnäytetyön tavoitteena oli luoda grafiikkamoottori, jolla voidaan esitellä yleisempiä tietokonegrafiikassa käytettyjä BRDF-malleja. Työssä luotu grafiikkamoottori suunniteltiin niin, että sillä on helppo valita eri BRDF-malleja tarkastelua varten kevyen käyttöliittymän avulla. Toisena päätavoitteena työssä oli tutustua BRDF-mallien taustalla olevaan matematiikkaan sekä BRDF-mallien toteuttamiseen näytönohjaimessa suoritettavilla varjostinohjelmilla.

Työn teoriaosassa kerrotaan BRDF-mallien teoriaa sekä niihin liittyvää matematiikkaa. Työn teoriaosassa on kerrottuna myös DirectX 11 -rajapintakokoelman teoriasta sekä sen sisältämien rajapintojen käyttötarkoituksista.

Työn lopputuloksena saatiin toimiva grafiikkamoottori, jolla voidaan esitellä reaaliaikaisessa grafiikassa käytettyjä BRDF-malleja. Työssä luotua ohjelmakoodia voidaan käyttää uudelleen muissa projekteissa, jotka liittyvät grafiikkaohjelmointiin.

Avainsanat: BRDF, tietokonegrafiikka, DirectX 11, HLSL, varjostin.

SEINÄJOKI UNIVERSITY OF APPLIED SCIENCES

Thesis abstract

Faculty: School of Technology

Degree programme: Information Technology

Specialisation: Software Engineering

Author: Teemu Niemi

Title of thesis: DirectX 11 graphics engine and BRDF-models

Supervisor: Petteri Mäkelä

Year: 2017 Number of pages: 48

The goal of this thesis was to create a graphics engine to demonstrate common BRDF-models used in real-time graphics. The design of the graphics engine was realized so that it allows user to easily switch from one BRDF-model to another.

One of the main goals was also to study the mathematics behind BRDF-models and how the models can be implemented in shader programs using the High Level Shading Language developed by Microsoft.

The theory part of this thesis consists of the theory behind BRDF-models that are demonstrated in this thesis. DirectX 11 is also described in the theory part of this thesis.

The result of this thesis was a graphics engine that allows users to examine the BRDF-models that are used in this thesis. The code that was written for the graphics engine can also be used in future graphics programming projects.

Keywords: BRDF, DirectX 11, HLSL, shader program, real-time graphics

SISÄLTÖ

Kuvioluettelo

Käytetyt termit ja lyhenteet

Anisotrooppinen	Ominaisuus, joka muuttuu eri suunnista mitattaessa.
BRDF	Bidirectional Reflectance Distribution Function eli suomennettuna kaksisuuntaisheijastusominaisuusfunktio.
BMP	Microsoftin kehittämä kuvaformaatti.
Geometriapuskuri	Joukko tietoa 3D-mallien geometriasta.
Hajavalo	Valo, joka heijastuu pinnasta jokaiseen suuntaan.
Heijastuskartta	Kuva, joka kertoo 3D-mallin pinnan heijastuvuudesta.
HLSL	High Level Shading Language on Microsoftin kehittämä ohjelmointikieli näytönohjaimiin.
Isotrooppinen	Ominaisuus, joka pysyy muuttumattomana mittaussuunnasta riippumatta.
JPEG	Joint Photographic Experts Group. Häviöllinen kuvaformaatti. Käytetään myös lyhennettä JPG.
Käyttöliittymä	Ohjelmassa olevat ikkunat, valikot ja valintaikkunat.
Metodi	Ohjelmoinnissa käytettävä termi, joka kuvaa toimintoa.
Ohjelmakirjasto	Kokoelma ohjelmia, luokkia tai aliohjelmia, joita voidaan käyttää apuna ohjelmoinnissa.
Olio	Sisältää loogisesti yhteenkuuluvaa ohjelman sisältämää tietoa ja toiminnallisuutta.
Peiliheijastus	Valo, joka heijastuu lähes kokonaan pinnasta tietystä kulmasta katsottaessa.
PNG	Portable Network Graphics. Pakkaamista tukeva häviötön kuvaformaatti.

Rajapinta	Mahdollistaa eri ohjelmien välisen kommunikoinnin.
Renderöinti	3D- tai 2D-kuvan muodostaminen tietokoneella.
TGA	Truevision Inc:in kehittämä kuvaformaatti, tunnetaan myös nimellä Targa.
Tekstuuri	Kuva, joka sijoitetaan 3D-mallin päälle.
Ympäristövalo	Valoa, joka ei tule suoraan valonlähteestä, vaan heijastuu ympärillä olevista pinnoista.
Varjostin	Varjostimet ovat ohjelmakoodia, jota tietokoneen näytönohjain suorittaa. Englanniksi varjostinta kutsutaan shaderiksi.
Varjostinmalli	Kertoo Microsoftin kehittämän HLSL-ohjelmointikielen version.

.

1 Johdanto

1.1 Työn tausta

Vaatimukset realistisen tietokonegrafiikan esittämiseen ovat kasvaneet nopeasti kehittyvien teknologioiden myötä. Realistisen grafiikan esittelyyn tarvitaan kuvaavia matemaattisia malleja, jotka takaavat katsojille animaatioissa ja peleissä mahdollisimman realistisen vaikutelman. Kyseessä olevia matemaattisia malleja kutsutaan BRDF-malleiksi. BRDF tulee englanninkielisestä termistä bi-directional reflectance distribution function. Suomeksi termi käännetään kaksisuuntaisheijastusominaisuusfunktioksi.

1.2 Työn tavoite

Opinnäytetyön tavoitteena on luoda grafiikkamoottori, jolla voidaan esitellä yleisempiä reaaliaikaisessa tietokonegrafiikassa käytettyjä BRDF-malleja. Työssä on myös tarkoitus perehtyä BRDF-mallien matematiikkaan.

1.3 Työn rakenne

Luvussa kaksi on esiteltynä Microsoftin DirectX 11 -ohjelmistorajapintakokoelma, jota käytetään työssä grafiikkamoottorin rakentamiseen. Luvussa kolme kerrotaan BRDF-mallien teoriaa sekä esitellään työssä käsiteltyjen mallien matematiikkaa. Luvussa neljä esitellään eri varjostinohjelmia. Luvussa viisi kuvataan ohjelman rakennetta ja toimintaa sekä ohjelmassa käytettyjä ohjelmakirjastoja. Luvussa kuusi on esiteltynä BRDF-mallien toteutus grafiikkamoottorissa. Luvussa seitsemän on yhteenveto.

2 Microsoft DirectX

Tässä luvussa kerrotaan Microsoftin DirectX-rajapintakokoelmasta, sen historiasta, sekä rajapintakokoelman sisällöstä. Lisäksi kerrotaan DirectX-kokoelman versioista ja opinnäytetyössä käytettävästä versiosta.

Vuonna 1995 ensimmäisen kerran Windows 95 -käyttöjärjestelmän ohella julkaistu DirectX on kokoelma Microsoftin kehittämiä kirjastoja, jotka mahdollistavat käskyjen lähettämisen suoraan audio- ja videolaitteisiin. DirectX sisältää useita eri kirjastoja, joiden avulla laitteistolle voidaan antaa käskyjä Windowsissa ja Xboxissa. (DirectX, [Viitattu 15.3.2017].)

DirectX-versiot julkaistaan yleensä yhdessä uusien Windows-versioiden kanssa, esimerkiksi Microsoft Windows Vistan kanssa julkaistiin DirectX 10, ja myöhemmin Windows 7- ja 8-järjestelmien kanssa julkaistiin DirectX 11. Uusin versio DirectX 12 julkaistiin Windows 10 -version yhteydessä.

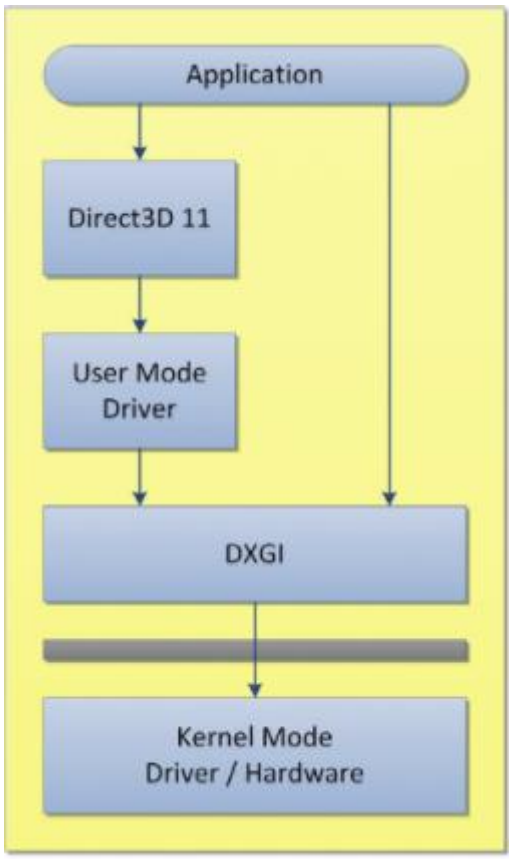
Opinnäytetyössä käytetty DirectX-versio on DirectX 11, joka on edelleen laajasti käytössä, vaikka nykyään on mahdollista käyttää DirectX 12 -versiota. DirectX 11 eroaa DirectX 12 -versiosta siten, että DirectX 12 sisältämät rajapinnat vaativat ohjelmoijalta syvää tuntemusta laitteiston toiminnasta (Eimandar 2013).

2.1 Direct3D

Direct3D on Microsoftin DirectX 11 mukana julkaistu natiivi ohjelmistorajapinta, joka lisää toiminnallisuuksia vanhojen Direct3D-versioiden päälle. Direct3D on käytettävissä Microsoft Vistassa sekä kaikissa Vistan jälkeen julkaistuissa Windows-versioissa. (Zink, Pettineo & Hoxley 2016.)

Direct3D-rajapintaa käytetään antamaan käskyjä ja kommunikoimaan suoraan tietokoneen grafiikkasuorittimen kanssa. Rajapintaa käytetään yleensä C/C++-ohjelmointikielillä, vaikka rajapintaa varten on luotu korkeamman tason kielille omia sovelluskehyksiä. (Zink, Pettineo & Hoxley 2016.)

Direct3D-rajapinnan alta löytyy useita sovelluspintoja, jotka eivät näy ohjelmalle. Nämä sovelluspinnat luovat Windowsin client-ympäristön grafiikka-arkkitehtuurin. (Zink, Pettineo & Hoxley 2016.)



Kuvio 1. Direct3D-rajapinnan kanssa käytettävät järjestelmät (Zink, Pettineo & Hoxley 2016).

Kuvion 1 määrittelemässä arkkitehtuurissa nähdään, että sovellus on kaikista korkeimmalla tasolla ja keskustelee pääasiassa Direct3D-rajapinnan kanssa. Alemmalla tasolla Direct3D keskustelee videolaitteiston user mode -ajurin kanssa, joka suorittaa sovelluksen antamia komentoja. Ajuri on tämän jälkeen vuorovaikutuksessa DXGI-sovelluskehyksen kanssa, joka on vastuussa matalan tason kommunikaatiosta kernel mode -ajurin ja saatavilla olevien laitteistoresurssien kanssa. (Zink, Pettineo & Hoxley 2016.)

2.2 XAudio2

XAudio2 on Microsoftin kehittämä matalan tason rajapinta signaalinkäsittelyyn ja äänen miksaukseen pelejä varten. XAudio2 on samankaltainen kuin sen edeltäjät DirectSound ja XAudio. Se on kehitetty korvaamaan DirectSound. (XAudio2, [Viitattu 15.3.2017].)

XAudio on tarkoitettu lähinnä pelejä varten. Microsoft kehoittaa dokumentaatiossaan käyttämään Media Foundation engineä, jos tarkoitus on tehdä tavallista musiikkia toistava sovellus. (XAudio2, [Viitattu 15.3.2017].)

XAudio2 tarjoaa matalalla vasteajalla toimivat työkalut äänitteiden muokkaamiseen. XAudio2 tarjoaa tuen myös dynaamisiin puskureihin, synkronoituun näytetarkkaan toistoon ja lähdemuunnoksiin. XAudio2-rajapintaa on mahdollista käyttää mahdollisimman pienellä määrällä ohjelmakoodia monimutkaisemmissakin järjestelmissä. (XAudio2, [Viitattu 15.3.2017].)

2.3 Xinput

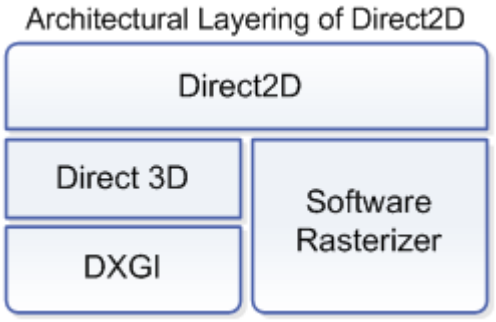
Xinput on uusi standardirajapinta, jota käytetään vastaanottamaan komennot näppäimistöltä tai Xbox 360 -ohjaimesta. Xinput on kehitetty korvaamaan Microsoftin DirectInput, mutta se ei tue vanhempia laitteita, jotka on tarkoitettu käytettäväksi DirectInputin kanssa. Jos Xinputia käyttävän ohjelman on tarkoitus tukea näitä vanhempia laitteita, pitää ohjelman käyttää myös DirectInputia. (Xinput, [Viitattu 16.3.2017].)

Xinput-rajapinnan avulla voidaan tunnistaa kaikki laitteeseen kytketyt Xbox 360 -ohjaimet. Xinput-rajapinnan avulla voidaan myös hallita Xbox 360 -ohjainten värinätoimintoja sekä vastaanottaa ja lähettää ääntä ohjaimiin yhdistetyistä kuulokemikrofoneista. (Xinput, [Viitattu 16.3.2017].)

2.4 Direct2D

Direct2D on Microsoftin kehittämä 2D-grafiikkaa varten luotu ohjelmistorajapinta. Direct2D-rajapinnan on tarkoitus auttaa luomaan tehokasta ja korkealaatuista 2D-grafiikkaa, kuten bittikarttoja ja tekstiä. Rajapinnan tarkoitus on toimia hyvin GDI-, GDI+ tai Direct3D-rajapintojen kanssa.

Direct2D on rakennettu alun perin käyttäen Direct3D 10.1 -rajapintaa, mutta Windows 8 julkaisun jälkeen Direct2D-rajapinta on rakennettu käyttämällä Direct3D 11.1 -rajapintaa. Direct2D-rajapinnan perustuminen Direct3D-rajapintaan tarkoittaa, että sovellukset, joissa käytetään Direct2D-rajapintaa hyötyvät myös Direct3D-rajapinnan tarjoamasta laitteistokiihdytyksestä. (Direct2D, [Viitattu 16.3.2017].)



Kuvio 2. Direct2D-arkkitehtuuri (Direct2D, [Viitattu 16.3.2017]).

Kuvio 2 esittelee Direct2D-arkkitehtuuria, jossa Direct2D sijoittuu Direct3D-rajapinnan päälle. Direct2D hyötyy Direct3D-rajapinnan kautta laitteistokiihdytyksestä. Tilanteissa, joissa laitteistokiihdytys ei ole mahdollinen, Direct2D hyötyy korkean suorituskyvyn ohjelmistorasteroijasta. (Direct2D, [Viitattu 16.3.2017].)

2.5 DXGI

Microsoft DirectX Graphics Infrastructure on sovelluskehys joka ottaa huomioon hitaammin kehittyvät grafiikan osa-alueet. DXGI-sovelluskehyksen päätavoite on

hallita matalan tason tehtäviä, jotka ovat erillisiä DirectX-rajapintakokoelman tehtävistä. Ensimmäinen komponentti, joka hyötyi DXGI-sovelluskehyksestä oli Microsoftin Direct3D 10. (DXGI, [Viitattu 18.3.2017].)

2.6 DirectCompute

Microsoftin DirectCompute mahdollistaa näytönohjaimen laskentatehon käyttämisen tavallisessa ohjelmoinnissa, mikä vaatii paljon rinnakkaislaskentatehoa. DirectComputea on tarkoitus hyödyntää silloin kun on paljon laskettavaa, jotta DirectComputella voidaan saavuttaa paras suorituskyky. (Thibieroz & Cebenoyan 2010.)

DirectComputea eli laskentavarjostinta voidaan tietokonegrafiikassa käyttää kuvan jälkiprosessointia varten, mutta sitä voidaan hyödyntää myös esimerkiksi fysiikan laskemisessa (Thibieroz & Cebenoyan 2010).

2.7 DirectX Diagnostics

DirectX Diagnostics on Microsoftin kehittämä työkalu vianmääritykseen. DirectX Diagnostics tunnistaa tietokoneeseen asennetut komponentit ja kertoo tietokoneeseen asennetun DirectX-version. (Heiskanen 2014.)

DirectX Diagnostics löytyy ohjelmana Windows-käyttöjärjestelmästä dxdiag-nimellä, sen avulla käyttäjä voi seurata tietokoneen laitteistoa ja ajureita. Ohjelman avulla voidaan tehdä tekstitiedosto, johon dxdiag-ohjelma listaa tietoa asennetuista ajureista ja käyttöjärjestelmästä. (Heiskanen 2014.)

2.8 DirectSetup

DirectSetup mahdollistaa DirectX-rajapintakokoelman asentamisen tietokoneelle. DirectSetup toimitetaan yleensä ohjelman mukana käyttäjille, jos ohjelma vaatii DirectX-rajapintakokoelmaan liittyviä tiedostoja. DirectSetupin uusimman version voi ladata Microsoftin verkkosivuilta. (Heiskanen 2014.)

3 BRDF-mallit

Tässä luvussa tutustutaan työssä käytettäviin erilaisiin BRDF-malleihin ja niiden matemaattisiin määritelmiin.

Tietokonegrafiikassa mahdollisimman realistisien kuvien luonnissa on pystyttävä selittämään valon vuorovaikutukset erilaisilla pinnoilla mahdollisimman tarkasti. Tällöin valon vuorovaikutusta erilaisilla pinnoilla voidaan selittää BRDF-malleilla. BRDF tulee sanasta Bidirectional Reflectance Distribution Function eli suomennettuna kaksisuuntaisheijastusominaisuusfunktio.

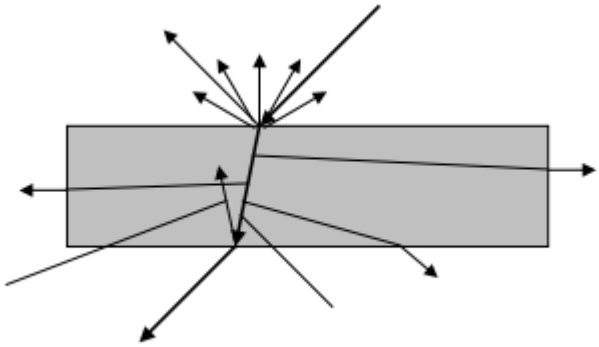
Viimeisen 35 vuoden aikana on esitelty monia analyyttisiä BRDF-malleja. Nämä mallit voidaan jakaa kahteen kategoriaan: empiiriset ja fyysisiin pohjautuvat mallit. Molempien kategorioiden mallit ovat vain arvioita erilaisten materiaalien heijastusominaisuuksista. Monet näistä malleista perustuvat materiaalien parametreihin, jotka ovat teoriassa mitattavissa, mutta käytännössä vaikeasti saatavilla. (Kurt & Edwards 2009.)

Vaihtoehtoisesti parametrien suoran mittaamisen sijaan voidaan käyttää goniospektrofotometriä, jolla saadaan oikeita näytteitä BRDF-funktioista, jonka jälkeen saatu data sijoitetaan valittuun analyyttiseen malliin käyttämällä erilaisia optimointitekniikoita. (Kurt & Edwards 2009.)

3.1 BRDF-funktioiden määritelmiä

Realistisien kuvien luonti perustuu numeeristen likiarvojen laskentaan kirkkausfunktiossa. Tämä merkitään kirjaimella L. Tietokonegrafiikan luonnissa L-funktion tärkeys perustuu faktaan, että sen avulla voidaan luoda yksinkertaisia ja tarkkoja malleja kuvan muodostumisesta ihmisen visuaalisessa järjestelmässä ja kameroissa. (Montes & Ureña 2012.)

Kirkkauteen vaikuttavat materiaalien eri ominaisuudet. Elektromagneettinen säteily on vuorovaikutuksessa hajautumisen, heijastumisen ja taittumisen kanssa. (Montes & Ureña 2012.)



.

Kuvio 3. Valon vuorovaikutukset (Wynn, [Viitattu 15.4.2017]).

Kuvio 3 esittelee valon vuorovaikutuksia valon ja aineen välillä. Aluksi valon osuessa aineeseen voi tapahtua kolme eri vuorovaikutusta: valon heijastuminen, valon imeytyminen ja valon lähettäminen. Osa valosta imeytyy väliaineessa. Koska valo on energiamuoto, energiaperiaatteen mukaan se on määritelty: valotapahtuma pinnalla = heijastunut valo + imeytynyt valo + lähetetty valo. (Wynn, [Viitattu 15.4.2017].)

Tavallinen funktiomerkintä BRDF-funktiosta

$$BRDF_{\lambda}(\theta i, \varphi i, \theta_o, \varphi_o, u, v) \quad (1)$$

missä muuttuja λ määrittelee BRDF-funktion olemaan riippuvainen valon aallonpituudesta. Parametrit θi, φi, edustavat tulevan valon suuntaa pallokoordinaateissa. Parametrit u ja v edustavat pinnan paikkaa tekstuuriavaruudessa. (Wynn, [Viitattu 15.4.2017].)

3.2 BRDF-luokat ja ominaisuuksia

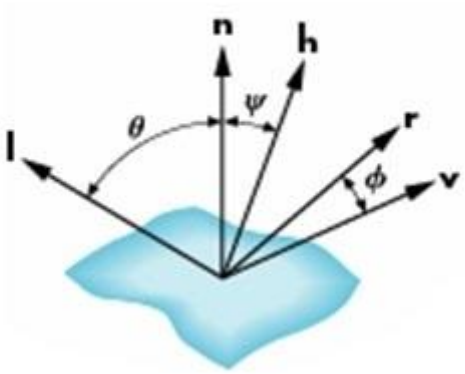
BRDF-funktiot voidaan jakaa kahteen eri luokkaan: suorat ja epäsuorat BRDF-funktiot. BRDF-funktioiden kaksi tärkeintä ominaisuutta ovat energian vastavuoroisuus ja säilyminen. (Wynn, [Viitattu 15.4.2017].)

Isotrooppiset eli suorat BRDF-funktiot edustaa sellaisia heijastumisen ominaisuuksia, jotka ovat muuttumattomia suhteessa pinnan rotaatioon. Tämä tarkoittaa sitä, jos katsoja ja valon lähde pidetään paikallaan, ja pintaa pyöritetään suhteessa sen normaaliin, niin pinnasta heijastuva valaistus ei muutu. Esimerkkinä tämänlaisesta materiaalista voidaan antaa sileä muovi, jolla on isotrooppinen BRDF. (Wynn, [Viitattu 15.4.2017].)

Anisotrooppiset eli epäsuorat BRDF-funktiot ovat funktioita, joissa pinnasta heijastuva valaistus muuttuu kun pintaa pyöritetään suhteessa sen normaaliin. Tämänlaisia materiaaleja ovat esimerkiksi hiottu metalli, satiini ja hiukset. Suurin osa oikean maailman BRDF-funktioista on jonkin verran anisotrooppisia. Isotrooppisuus määritelmänä on hyödyllinen kuitenkin, koska moni analyyttinen BRDF-malli sijoittuu isotrooppiseen luokkaan. (Wynn, [Viitattu 15.4.2017].)

3.3 BRDF-malleja

BRDF-malleja on useita, kaikki niistä eivät sovellu tietokonegrafiikassa käytettäväksi. Seuraavassa esitellään muutamia yleisimpiä malleja, joita käytetään tietokonegrafiikassa.



Kuvio 4. Puolivälivektorin havainnollistaminen (Shading2, 2016).

Kuviossa 4 havainnollistetaan monessa BRDF-mallissa käytettyä puolivälivektoria H tai h. Kuvasta nähdään, että puolivälivektori sijaitsee puolessa välissä vektoreita v ja l, jotka ovat pinnasta kameraan ja pinnasta valoon suuntautuvat vektorit.

$$h = \frac{l + v}{|l + v|} \tag{2}$$

Kaavassa (2) on esiteltynä kuviossa 4 havainnollistettu puolivälivektorin h laskukaava, missä v on kappaleesta kameraan menevä vektori ja l on kappaleesta valon lähteeseen kulkeva vektori.

3.3.1 Phongin BRDF

Vuonna 1973, Bui Tuong Phong esitteli oman BRDF-mallinsa, mikä kuvaa hyvin valon heijastumista sileistä ja muovisista pinnoista. Phongin BRDF-malli on empiirinen BRDF-malli, sillä se ei seuraa tarkasti fysiikan lakeja. (Montes & Ureña 2012.)

Phongin BRDF-malli saadaan yhtälöstä

$$k_d f_{lambert} + k_s * (V \cdot R)^n \quad (3)$$

missä V on kameran ja kappaleen välinen vektori ja n on vapaasti määriteltävä parametri, joka vaikuttaa heijastukseen. Kaavassa (3) $k_s * (V \cdot R)^n$ kuvaa peiliheijastusta ja $k_d f_{lambert}$ kuvaa Lambertian hajavalon kaavaa eli $\frac{\rho}{\pi}$, missä ρ on pinnan heijastuvuus (Montes & Ureña 2012).

$$R = 2(N \cdot L)N - L \quad (4)$$

Kaava (4) kuvaa valon heijastumissuuntaa, jossa vektori L on valon suunta osumakohdassa ja N on pinnan normaalivektori (Montes & Ureña 2012).

3.3.2 Blinn-Phongin BRDF

Blinn-Phongin BRDF-malli on yleisin valaistuksen laskemisessa käytetty BRDF-malli. Blinnin malli esittää, että käyttämällä puolivälivektoria H Phongin kaavassa (3) korvaamalla kaavan (3) heijastusvektori R vähennetään laskutehtävien määrää. Tämä mahdollistuu sillä, että ei tarvitse enää löytää heijastusvektoria. (Montes & Ureña 2012.)

Blinn-Phongin BRDF-malli määritellään kaavalla

$$k_d + k_s(N \cdot H)^n \quad (5)$$

missä N on pinnan normaali, H on puolivälivektori ja n on heijastuskerroin. (Montes & Ureña 2012.)

3.3.3 Cook-Torrancen BRDF

Cook-Torrancen BRDF-mallissa oletetaan, että kappaleen pinnassa olevat mikrofaktit, jotka on suunnattu puolivälivektoriin h, muodostavat lopullisen heijastuksen. Puolivälivektori h on esitelty kaavassa (2). Cook-Torrancen BRDF-malli esittelee myös uuden tyyppisen materiaalin tietokonegrafiikassa. Se jakaa metalliset ja ei-metalliset materiaalit. (Montes & Ureña 2012).

Cook-Torrancen BRDF-malli määritellään seuraavasti

$$f_r = k_d f_{lambert} + k_s f_{cook-torrance} \tag{6}$$

$k_d f_{lambert}$ määrittelee hajavalon ja $k_s f_{cook-torrance}$ määrittelee peiliheijastuksen. Yhtälön $k_s f_{cook-torrance}$ saadaan laskettua seuraavasti:

$$f_{cook-torrance} = \frac{DFG}{4(w_0 \cdot n)(w_1 \cdot n)} \tag{7}$$

Kaavassa (7) muuttuja D on normaalin hajautuminen, mikä kertoo pinnan mikrofaktien suuntauksen puolivektorin H mukaan. Tähän vaikuttaa pinnan karheus. Yhtälössä D saadaan laskettua Trowbridge-Reitzin GGX-funktiolla:

$$NDF_{GGXTR}(n, h, \alpha) = \frac{\alpha^2}{\pi(n \cdot h)^2(\alpha^2 - 1) + 1)^2} \tag{8}$$

kaavassa (8) parametreinä on pinnan normaalivektori n, puolivälivektori h ja vakio a, mikä on pinnan karheus. (Vries 2017.)

Kaavassa (7) G kuvaa geometriafunktiota, joka kertoo mikrofaktien synnyttämien varjojen vaikutuksen pinnasta heijastuvaan valoon verrattaen karheilla pinnoilla. Yhtälön G saadaan laskettua Schlick-Beckmannin Schlick-GGX -funktiosta

$$G_{SchlickGGX}(n, v, k) = \frac{n \cdot v}{(n \cdot v)(1 - k) + k} \tag{9}$$

missä

$$k = \frac{(\alpha + 1)^2}{8} \tag{10}$$

Yhtälön F kuvaa Fresnelin yhtälöä, mikä taas kuvaa heijastuvan ja taittuvan valon suhdetta. Tämä muuttuu sen kulman mukaan, josta pintaa katsotaan. F saadaan laskettua seuraavanlaisesti

$$F_{Schlick(n,v,F_0)} = F_0 + (1 - F_0)\big(1 - (n \cdot v)\big)^5 \tag{11}$$

Yhtälön F_0 on yleensä joku ennalta määrätty vakio. (Vries 2017.)

3.3.4 Wardin BRDF

Wardin BRDF-malli on laskennallisesti kevyt BRDF-malli kuvaamaan anisotrooppista heijastusta.

Wardin isotrooppisen BRDF-mallin matemaattinen määritelmä on

$$F_{r,iso}(w_0, w_i) = \frac{k_s}{\sqrt{\cos(w_0)\cos(w_i)}} \frac{\exp\left[-tan^2\left(\frac{(h \cdot n)}{\alpha^2{}_m}\right)\right]}{4\pi\alpha^2{}_m} \tag{12}$$

Wardin anisotrooppisen BRDF-mallin määritelmä on

$$F_{r,ani}(w_0, w_i) = \frac{k_s}{4\pi\alpha_x\alpha_y\sqrt{\cos(w_0)\cos(w_i)}} e^{\left[-\frac{\left[\frac{(h\cdot x)}{\alpha_x}\right]^2 + \left[\frac{(h\cdot y)}{\alpha_y}\right]^2}{(h\cdot n)^2}\right]} \tag{13}$$

missä

$$(h \cdot x) = \frac{\sin(\theta wo)\cos(\varphi wo) + \sin(\theta wi)\cos(\theta wi)}{|\,h\,|} \tag{14}$$

$$(h \cdot y) = \frac{\sin(\theta \mathrm{wo})\sin(\varphi \mathrm{wo}) + \sin(\theta \mathrm{wi})\sin(\varphi \mathrm{wi})}{|\,h\,|} \tag{15}$$

ja

$$(h \cdot n) = \frac{\cos(\theta wo) + \cos(\theta wi)}{|h|} \quad (16)$$

Wardin BRDF-malli kuuluu kokeellisiin BRDF-malleihin, joten se on mitattu luonnosta goniospektrofotometrillä. (Montes & Ureña 2012.)

3.3.5 Lambertianin BRDF

Lambertianin BRDF-malli kuvailee pintaa, joka heijastaa valoa joka suuntaan saman verran. Oikeassa maailmassa ei ole pintoja, jotka noudattavat tätä BRDF-mallia, mutta mattapinta on lähin esimerkki tämänlaisesta pinnasta. (Montes & Ureña 2012.)

Lambertianin BRDF-malli saadaan kaavasta

$$\frac{\rho}{\pi} \quad (17)$$

missä ρ on pinnan heijastuvuus. Tätä BRDF-mallia käytetään yleensä laskemaan kappaleen pinnasta heijastuvaa hajavaloa, joka yhdistetään erikseen peiliheijastusta kuvaavaan kaavaan. (Montes & Ureña 2012.)

3.3.6 Oren-Nayarin BRDF

Oren-Nayarin BRDF-malli on paranneltu versio klassisesta Lambertianin BRDF-mallista. Oren-Nayarin BRDF-malli pystyy selittämään valon heijastumisen mattapinnoilta geometriaoptiikalla. Tästä syystä malli luetaan fyysisiin tai teoreettisiin BRDF-malleihin. (Montes & Ureña 2012.)

Oren-Nayarin mallilla mallinnetaan hajavalon osuutta valoyhtälössä. Oren-Nayarin BRDF-mallin matemaattinen määritelmä on

$$\mathrm{fr}(\mathrm{wo},\mathrm{wi}) = \frac{\rho}{\pi}\Big(\mathrm{A} + \mathrm{B}\,\max\big(0,\cos(\varphi\mathrm{wi} - \varphi\mathrm{wo})\big)\sin(\mathrm{a})\tan(\mathrm{b})\Big) \tag{16}$$

missä

$$\mathrm{a} = \max(\theta\mathrm{wo},\theta\mathrm{wi}), \mathrm{b} = \min(\theta\mathrm{wo},\theta\mathrm{wi}) \tag{17}$$

ja

$$A = 1 - 0.5\frac{{\alpha^2}_m}{{\alpha^2}_m + 0.33} \tag{18}$$

ja

$$B = 0.45\frac{{\alpha^2}_m}{{\alpha^2}_m + 0.09} \tag{19}$$

Oren-Nayarin BRDF-malli ottaa huomioon myös energian säilymislain. (Montes & Ureña 2012.)

4 Varjostimet ja HLSL

Tässä luvussa kerrotaan, mitä ovat varjostimet eli shaderit, sekä kuinka niitä luodaan käyttämällä HLSL-ohjelmointikieltä.

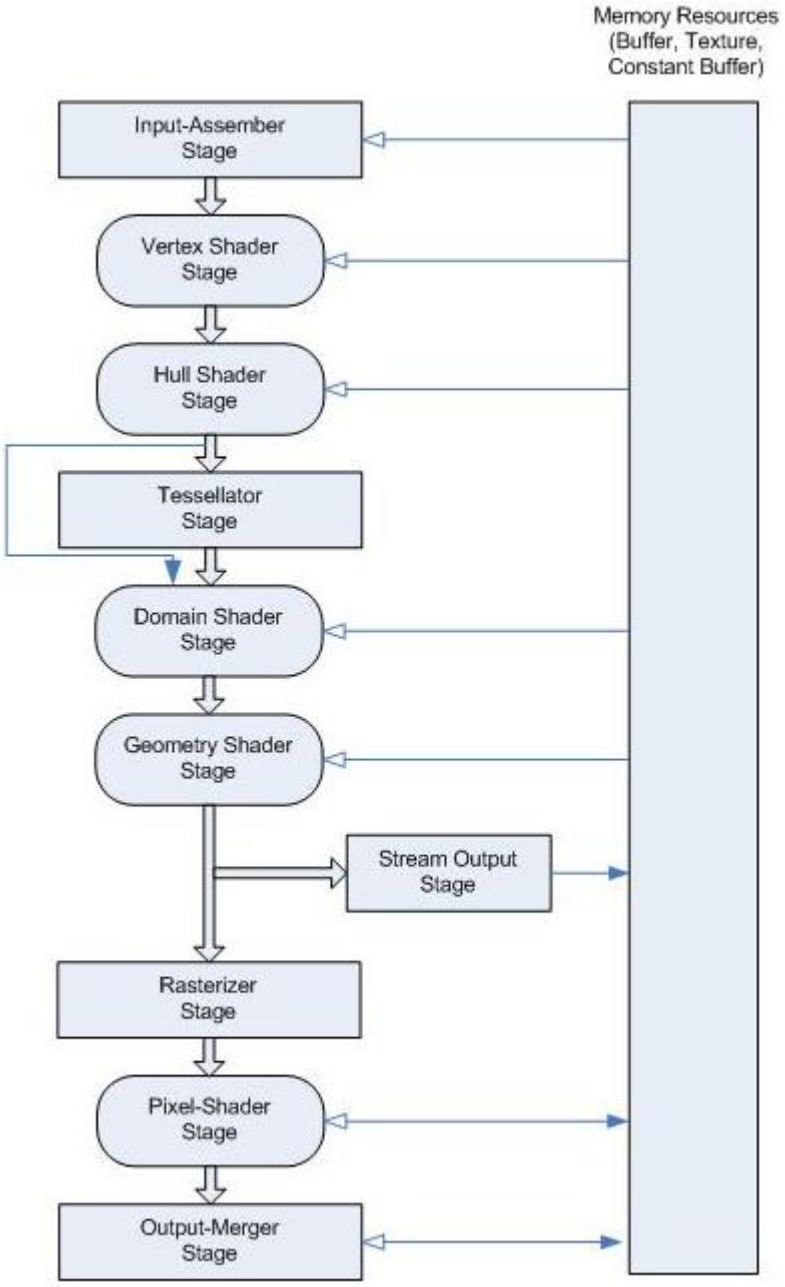
Modernissa grafiikkaohjelmoinnissa on tapana hyödyntää uusien näytönohjaimien tarjoamaa grafiikkakiihdytystä.

> Grafiikkakiihdytystä käyttävät operaatiot ohjelmoidaan erityisillä varjostinohjelmilla, jotka suoritetaan näytönohjaimella. Vaikka varjostinohjelmia käyttämällä onkin mahdollista toteuttaa laajalti erilaisia tehosteita, on varjostinohjelmia käyttävässä ohjelmassa huolehdittava oikeellisesta varjostinten alustuksesta ja piirtoon käytettävän datan syöttämisestä. (Korhonen 2013.)

HLSL High Level Shading Language on Microsoftin kehittämä ohjelmointikieli varjostimia varten. HLSL-kielellä voidaan kirjoittaa C-tyylisiä ohjelmia näytönohjaimen suoritettavaksi. HLSL-ohjelmointikieli julkaistiin ensimmäisen kerran DirectX 9 -rajapintakokoelman mukana. (Shader Models, [Viitattu 3.4.2017].)

Vuosien saatossa HLSL-kielestä on julkaistu uusia varjostinmalleja, jotka tuovat uusia ominaisuuksia vanhojen lisäksi. Alussa varjostinmalli tuotiin DirectX 8 -rajapintakokoelmaan ja siinä oli Assembly-tason ja C-tyylisiä komentoja, mutta tällä mallilla oli paljon rajoituksia johtuen sen aikaisesta grafiikkalaitteistosta. Myöhemmin varjostinmallit 2 ja 3 lisäsivät huomattavasti komentojen määrää. Ne olivat alkuperäistä varjostinmallia 1 tehokkaampia. Varjostinmalleissa 2 ja 3 oli silti ensimmäisen varjostinmallin mukana tulleita rajoitteita. (Shader Models, [Viitattu 3.4.2017].)

Windows Vistasta alkaen varjostinmalli 4 oli täysin uudelleen suunniteltu ja se mahdollisti lähes rajattoman määrän komentoja. Ainoana rajoitteena oli grafiikkalaitteiston kapasiteetti. (Shader Models, [Viitattu 3.4.2017].)



Kuvio 5. Direct3D 11 -rajapinnan grafiikkakanava
(Direct3D pipeline, [Viitattu 1.4.2017]).

Kuvio 5 kertoo Direct3D 11 -rajapinnan sisältämän ohjelmoitavan kanavan. Direct3D 11 -rajapinnan grafiikkakanavaa ohjelmoidaan varjostinohjelmilla. Kuvio 5 näyttää, missä järjestyksessä kukin varjostinohjelma suoritetaan. (Direct3D pipeline, [Viitattu 1.4.2017].)

Input-Assembler-vaiheessa input assembler käsittelee käyttäjän täyttämät puskurit ja kokoaa ne sitten muiden kanavan vaiheiden käsiteltäväksi. Data voi olla pisteitä, viivoja tai kolmioita. (Direct3D pipeline, [Viitattu 1.4.2017].)

Verteksivarjostin-tasolla verteksivarjostin prosessoi kulmapistedatan, minkä se saa input assemblerista. Tämän jälkeen data lähetetään Tesselaatio-tasolle, missä vähän yksityiskohtia sisältävät osapinnat voidaan muuttaa laadukkaammiksi. Tesselaatio-taso on hyödyllinen kun käsitellään yksinkertaisia malleja ja niistä halutaan laadukkaamman näköisiä. (Direct3D pipeline, [Viitattu 1.4.2017].)

Geometriavarjostin-tasolla käsitellään kokonaisia kolmioita, viivoja tai yksittäisiä kulmapisteitä. Geometriavarjostin siis mahdollistaa useamman kulmapisteen käsittelyn toisin kuin verteksivarjostin. (Direct3D pipeline, [Viitattu 1.4.2017].)

Stream-Output-tasolla dataa lähetetään geometriavarjostintasolta yhteen tai useampaan muistipuskuriin. Geometriavarjostintason ollessa pois käytöstä, dataa lähetetään verteksivarjostintasolta. Tätä dataa voidaan sen jälkeen käsitellä prosessorilla. (Direct3D pipeline, [Viitattu 1.4.2017].)

Rasterointi-tasolla kulmapistedata ja muodot muutetaan rasterikuvaksi, joka muodostuu pikseleistä. Rasterointitason tarkoitus on muuttaa 3D-data näkymään 2D-kuvana näytöllä. (Direct3D pipeline, [Viitattu 1.4.2017].)

Pikselivarjostin-tasolla voidaan muokata yksittäisten pikselien dataa, se mahdollistaa kuvan jälkikäsittelyn sekä valaistuksen laskemisen. Pikselivarjostintaso on mahdollista ottaa pois käytöstä, jos sitä ei tarvita. (Direct3D pipeline, [Viitattu 1.4.2017].)

Output-Merger-taso on viimeinen taso, joka määrittelee pikselin lopullisen värin. Output-Merger käsittelee usean eri grafiikkakanavan dataa. (Direct3D pipeline, [Viitattu 1.4.2017].)

Verteksivarjostinohjelman tarkoitus on tehdä käyttäjän määrittelemiä laskutoimituksia jokaiselle monikulmion kärkipisteelle hyödyntämällä näytönohjaimen rinnakkaista laskentaa (Korhonen 2013).

Geometriavarjostimessa lasketaan primitiivikohtaisia laskutoimituksia. Geometriavarjostimella voidaan tehdä muutoksia valmiiksi luodun kappaleen muotoon. (Korhonen 2013.)

Pikselivarjostin käsittelee pikselikohtaista dataa verteksivarjostimen ja geometriavarjostimen ulostulojen perusteella. Pikselivarjostimessa lasketaan pääosin kappaleiden valaistus ja kuvan jälkiprosessointi. Pikselivarjostin lasketaan jokaiselle ohjelmalle varatulle pikselille myös hyödyntämällä näytönohjaimen rinnakkaista laskentaa. (Korhonen 2013.)

Laskentavarjostimet lasketaan täysin muiden varjostinohjelmien ulkopuolella. Eikä laskentavarjostin ole muista varjostinohjelmista riippuvainen, joten laskentavarjostimen toiminta on itsenäistä. Laskentavarjostimen avulla voidaan hyödyntää näytönohjaimen rinnakkaista laskentaa muutenkin kuin pelkästään graafisesti. Laskentavarjostinteknologia tunnetaan myös nimeltä DirectCompute-teknologia. (Korhonen 2013.)

5 Ohjelman rakenne ja toiminta

Ohjelmassa esitellään yleisempiä reaaliaikaisessa grafiikassa käytettyjä BRDF-malleja. Ohjelmassa pystytään kätevästi vaihtamaan ajon aikana eri BRDF-mallien välillä graafisen käyttöliittymän avulla. Ohjelma noudattaa olio-ohjelmoinnin sääntöjä, eli ohjelman jokaisesta osasta on tehty oma luokkansa. Ohjelmassa käytetään 3D-malleissa pääasiassa wavefront formaattia, joka on yleinen 3D-mallien tallennusformaatti. Ohjelma pystyy lukemaan muitakin 3D-formaatteja, mutta niiden toimiminen oikein ei ole taattua. Ohjelmassa kuvien lataaminen ohjelman muistiin tapahtuu FreeImage-nimisen ohjelmakirjaston avulla. FreeImage tukee lähes kaikkia yleisimmin käytettyjä kuvaformaatteja, kuten PNG, JPEG, BMP ja TGA.

Grafiikkamoottorissa kuvan muodostaminen on viivästetty eli kuva muodostetaan kahdessa osassa. Englanniksi tätä tapaa kutsutaan Deferred Rendering. Kuvan muodostaminen tapahtuu niin, että ensimmäisessä vaiheessa muodostetaan ohjelmasta riippuen noin 4 kuvaa, joihin tallennetaan nähdyn alueen kappaleiden paikat, värit, pintojen normaalit sekä yleensä heijastuskartat. Tämän vaiheen jälkeen nämä 4 kuvaa annetaan pikselivarjostimelle valaistuksen laskemista varten. Tämä tapa mahdollistaa huomattavan määrän enemmän valoja, sillä valaistus lasketaan vain nähdylle alueelle.

5.1 Ohjelman vaiheet

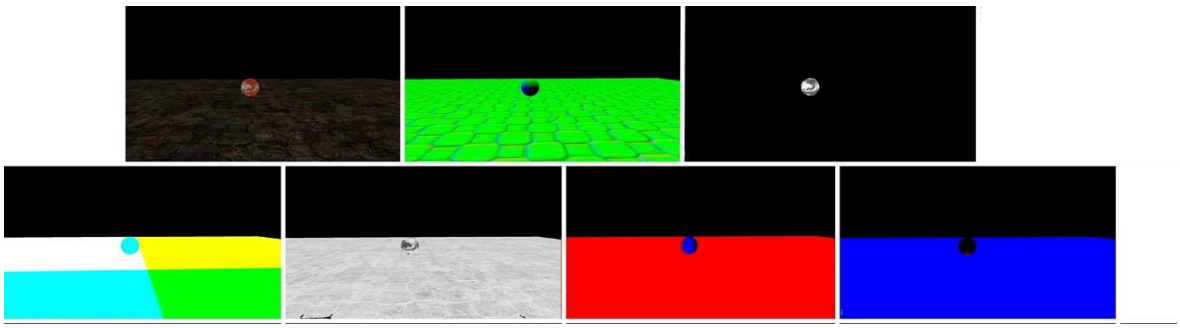
Ohjelmassa on kolme vaihetta, jotka ovat aloitus, piirtämissilmukka ja lopetus. Ohjelman alussa valmistellaan ja luodaan ikkuna. Tämän jälkeen ladataan 3D-malleja muistiin sekä ladataan ja käännetään kaikki varjostintiedostot. Ohjelman alussa valmistellaan myös Direct3D-olio, joka vastaa grafiikkalaitteiston käytöstä ja varjostimien kääntämisestä.

Piirtämissilmukassa suoritetaan varsinainen 3D-grafiikan näyttäminen useassa eri vaiheessa. Piirtämissilmukan alussa päivitetään projektio- ja kameramatriisi käyttäjän toimintojen mukaan. Matriisien päivittämisen jälkeen kootaan geometriapuskuri, jossa kaikki kappaleet kamerasta nähden piirretään seitsemään

eri tekstuuriin. Geometriapuskurin sisältö on tämän vaiheen jälkeen seuraavanlainen: väri, pinnan normaalit, heijastukset, paikka, karheus, pinnan tangentit ja pinnan bitangentit.

5.2 Viivästetty kuvan piirto

Viivästetyssä kuvan piirtämisessä hyödynnetään tekstuurien värikanavia tallentamalla niihin vektoreita. Tämä kuitenkin vaatii vähintään 16-bittisten värikanavien käyttöä, jotta tallennettujen vektoreiden tarkkuus säilyy. Näytölle piirrettäessä on kuitenkin mahdollista käyttää vain 8-bittisiä värikanavia.



Kuvio 6. Geometriapuskurin sisältö

Kuviossa 6 nähdään ruutukaappauksina otetut tekstuurit geometriapuskurissa. Näitä käytetään valaistuksen laskemisen apuna. Kuviossa on järjestyksessä vasemmalta oikealle: värit, pintojen normaalit, heijastuskartta, kappaleiden paikat XYZ-avaruudessa, karheus, tangentit ja bitangentit.

5.3 3D-mallien lataaminen

Ohjelmassa käytetään 3D-mallien lataamiseen ohjelmakirjastoa nimeltä Assimp. Assimp mahdollistaa monimutkaistenkin 3D-mallien käyttämisen helposti ja ohjelmakirjasto tukee monia eri 3D-mallien tallennusformaatteja. (Assimp, [Viitattu 21.5.2017].)

Assimp-ohjelmakirjasto on tarkoitettu käytettäväksi C/C++-ohjelmointikielien kanssa, mutta kirjastoa on mahdollista käyttää myös Pythonilla, D:llä ja Blitzmax ohjelmointikielillä. Ohjelmakirjaston verkkosivujen mukaan tulossa on tuki myös Javalle sekä C#-kielelle. (Assimp, [Viitattu 21.5.2017].)

Kuvio 7. Crytekin luoma 3D-malli Dubrovnikin Sponza-palatsista

Kuviossa 7 esitellään Crytekin luomaa 3D-mallia Sponza-nimisestä palatsista. Tämä 3D-malli on erittäin suosittu grafiikkaohjelmoijien kesken, kun testataan globaalia valaistusta tai varjojen luontia. Tämä 3D-malli ei kuitenkaan sovellu tämän opinnäytetyön aihepiiriin esittelemään BRDF-malleja, sillä mallissa on useita erilaisia materiaaleja, joihin on haastavaa valita yhtä sopivaa BRDF-mallia. Ohjelmassa käytetty Assimp-ohjelmakirjasto mahdollistaa helposti tämän kaltaisen monimutkaisen mallin käyttämisen ohjelmassa.

5.4 Ohjelman käyttöliittymä

Käyttöliittymä on ohjelmassa toteutettu AntTweakBar-nimisen ohjelmakirjaston avulla. Käyttöliittymän on tarkoitus olla kevyt ja mahdollistaa vain muutama toiminto, mikä helpottaa BRDF-mallien tarkastelua ja 3D-mallien sekä valojen liikuttamista.

Kuvio 8. Ohjelman käyttöliittymä

Kuviossa 8 on esitettynä ohjelman käyttöliittymä, jossa vasemmalla on valintaruudut BRDF-malleille, ja valon paikan sekä värin muuttamista varten olevat valintaruudut. Oikealla näkyvässä ikkunassa on kappaleiden paikan, koon ja kierron muuttamista varten olevat valintaruudut.

6 BRDF-mallien toteutus

Tässä kappaleessa kerrotaan BRDF-mallien toteutuksesta opinnäytetyötä varten tehdyssä grafiikkamoottorissa. Työhön valittiin sellaisia BRDF-malleja, joita käytetään yleisesti tietokonegrafiikassa, ja joihin löytyi tarpeeksi ohjeistusta toteutusta varten.

Kaikissa BRDF-malleissa, jotka laskevat peiliheijastuksen, käytetään valoyhtälössä Lambertianin BRDF-mallia. Lambertianin mallin ja Oren-Nayarin mallin kohdalla lasketaan vain hajavalo eikä yhtälöön oteta mukaan peiliheijastusta. Kaikkiin yhtälöihin lisätään vielä vakio globalAmbient eli ympäristövalo.

6.1 Phongin mallin toteutus

Phongin mallin toteutus grafiikkamoottorissa on yksinkertainen, eikä se vaadi montaa riviä varjostinkoodia. Phongin BRDF-malli on erittäin yleinen varsinkin netistä löydettävissä aloittelijan-oppaissa, joissa opetellaan käyttämään joko OpenGL-rajapintaa tai Direct3D-rajapintaa.

```
float3 Phong(float3 N, float3 V, float3 L, float a, float3 albedo, float metallic, float3 radiance)
{
    float NoL = dot(N, L);
    float3 F0 = 0.03f.xxx;
    float3 metallic1 = float3(metallic, metallic, metallic);

    F0 = lerp(F0, albedo, metallic1);


    float3 diffuse = DiffuseLambertian(albedo);

    float3 reflection = reflect(-L, N);

    float3 specular = pow(max(dot(N, reflection), 0.0), 15.0);



    return diffuse + (specular * F0) * radiance * NoL;

}
```

Kuvio 9. Phongin BRDF-mallin HLSL-koodi

Kuvion 9 HLSL-koodissa Phongin BRDF-malli lasketaan niin, että hajavaloon käytetään Lambertianin BRDF-mallia ja peiliheijastusta kuvaa Phongin BRDF-malli. Tähän lisätään vielä vakio globalAmbient kuvaamaan ympäristön valoa. Täten kuviossa 9 esitelty metodi toteuttaa yhtälön (3).

Kuvio 10. Phongin mallin toteutus ohjelmassa

Kuviossa 10 esitelty Phongin heijastus on paras muovisille ja sileille pinnoille, se ei sovellu kuvassa olevalle osittain ruosteiselle ja karhealle materiaalille.

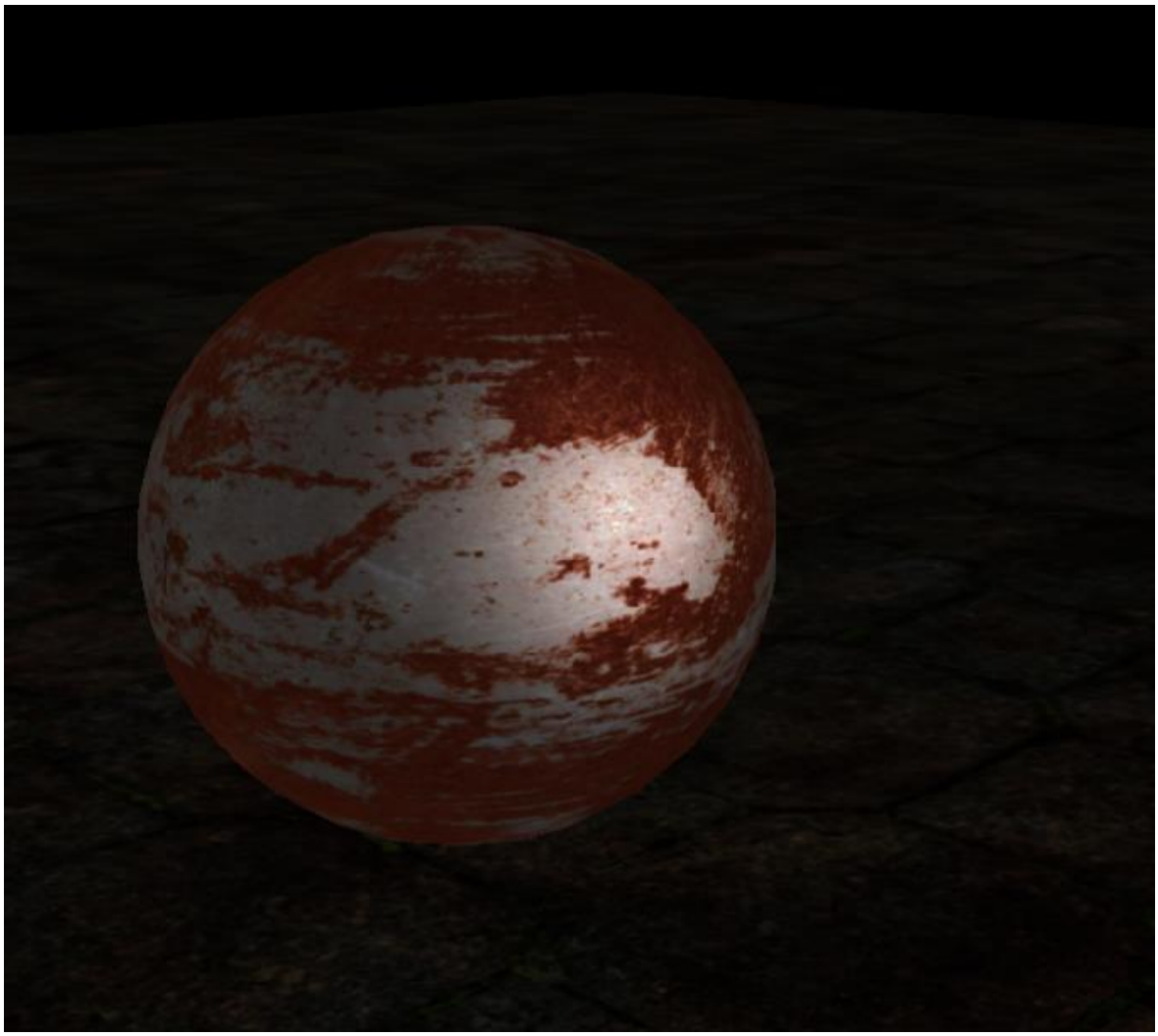
6.2 Blinn-Phongin mallin toteutus

Blinn-Phongin mallissa peiliheijastus lasketaan yhtälön (5) mukaan.

```
float3 BlinnPhong(float3 N, float3 V, float3 L, float a, float3 albedo, float metallic, float3 radiance)
{
    float NoL = dot(N, L);
    float3 H = normalize(V + L);
    float3 F0 = 0.03f.xxx;
    float3 metallic1 = float3(metallic, metallic, metallic);

    F0 = lerp(F0, albedo, metallic1);

    float3 diffuse = DiffuseLambertian(albedo);



    float3 specular = pow(max(dot(N, H), 0.0), 5.0);



    return diffuse + (specular * F0) * radiance * NoL;

}
```

Kuvio 11. Blinn-Phongin HLSL-koodi

Kuviossa 11 esitellään Blinn-Phongin BRDF-mallin toteutus HLSL-ohjelmointikielellä. Blinn-Phongin BRDF-malli toteutetaan muuten samalla tavalla kuin Phongin malli, mutta Blinn-Phong käyttää puolivälivektoria H heijastusvektorin tilalla. Tämä on laskennallisesti kevyempi toteuttaa, koska ei tarvitse etsiä heijastusvektoria R.



Kuvio 12. Blinn-Phongin mallin toteutus ohjelmassa

Kuviossa 12 nähdään, kuinka Blinn-Phongin mallissa kappaleesta heijastuva peiliheijastus on himmeämpi kuin Phongin mallissa.

6.3 Cook-Torrancen mallin toteutus

Cook-Torrancen BRDF-malli on ehkä yleisin käytetty BRDF-malli laskemaan peiliheijastusta realistisuuteen tähtäävässä tietokonegrafiikassa. Cook-Torrancen malli on laskennallisesti jo hiukan vaativampi, mutta lopputulos on huomattavasti parempi kuin aiemmin esitellyissä Blinn-Phongin ja Phongin malleissa.

```
float3 CookTorrance(float3 N, float3 V, float3 L, float a, float3 albedo, float metallic, float3 radiance)
{
    float3 H = normalize(V + L);
    float3 F0 = 0.03f.xxx;
    float3 Cook = 0.0f;

    float3 metallic1 = float3(metallic, metallic, metallic);

    F0 = lerp(F0, albedo, metallic1);


    float NoH = dot(N, H);
    float NoV = dot(N, V);
    float NoL = dot(N, L);
    float VoH = dot(V, H);
    float HoV = dot(H, V);


    float NDF = DistributionGGX(N, H, a);
    float G   = GeometrySmith(N, V, L, a);
    float3 F  = FresnelSchlick1(max(HoV,0.0), F0);

    float3 kS = F;
    float kD = float3(1.0f, 1.0f, 1.0f) - kS;
    kD *= 1.0 - metallic1;

    float3 nominator = NDF * G * F;
    float denom = 4 * max(dot(N, V), 0.0) * max(dot(N, L), 0.0) + 0.001;
    float3 specular = nominator / denom;

    NoL = saturate(NoL);
    float3 diffuseC = DiffuseLambertian(albedo);
    float3 specularC = specular;

    Cook = (kD * diffuseC + specularC) * radiance * NoL;

    Cook = Cook / (Cook + 1.0f);
    Cook = pow(Cook, 1.0f / 2.2f);

    return Cook;

}
```

Kuvio 13. Cook-Torrancen BRDF-mallin HLSL-koodi

Kuviossa 13 on esitettynä kokonaisuudessaan Cook-Torrancen BRDF-mallin HLSL-koodi ja se toteuttaa yhtälön (6). Metodissa haetaan normaalin hajautuminen NDF metodista nimeltä DistributionGGX, mikä toteuttaa yhtälön (8). Tässä käytetty yhtälö on nimeltään jo aiemmin mainittu Trowbridge-Reitz GGX. Geometriafunktio saadaan Smithin metodista. Fresnelin yhtälö lasketaan Fresnel-Schlickin

likiarvosta. Lopussa laskettuun Cook-Torrancen peiliheijastukseen lisätään Lambertianin mallista hajavalon osuus.

```
float DistributionGGX(float3 N, float3 H, float roughness)
{
    float a = roughness * roughness;
    float a2 = a * a;
    float NdotH = max(dot(N, H), 0.0);
    float NdotH2 = NdotH * NdotH;

    float nom = a2;
    float denom = (NdotH2 * (a2 - 1.0) + 1.0);
    denom = PI * denom * denom;

    return nom / denom;
}

float GeometrySchlickGGX(float NdotV, float roughness)
{
    float r = (roughness + 1.0);
    float k = (r * r) / 8.0;

    float nom = NdotV;
    float denom = NdotV * (1.0 - k) + k;

    return nom / denom;
}
float GeometrySmith(float3 N, float3 V, float3 L, float roughness)
{
    float NdotV = max(dot(N, V), 0.0);
    float NdotL = max(dot(N, L), 0.0);
    float ggx2 = GeometrySchlickGGX(NdotV, roughness);
    float ggx1 = GeometrySchlickGGX(NdotL, roughness);

    return ggx1 * ggx2;
}
```

Kuvio 14. Cook-Torrancen varjostimessa käytettyjä metodeja

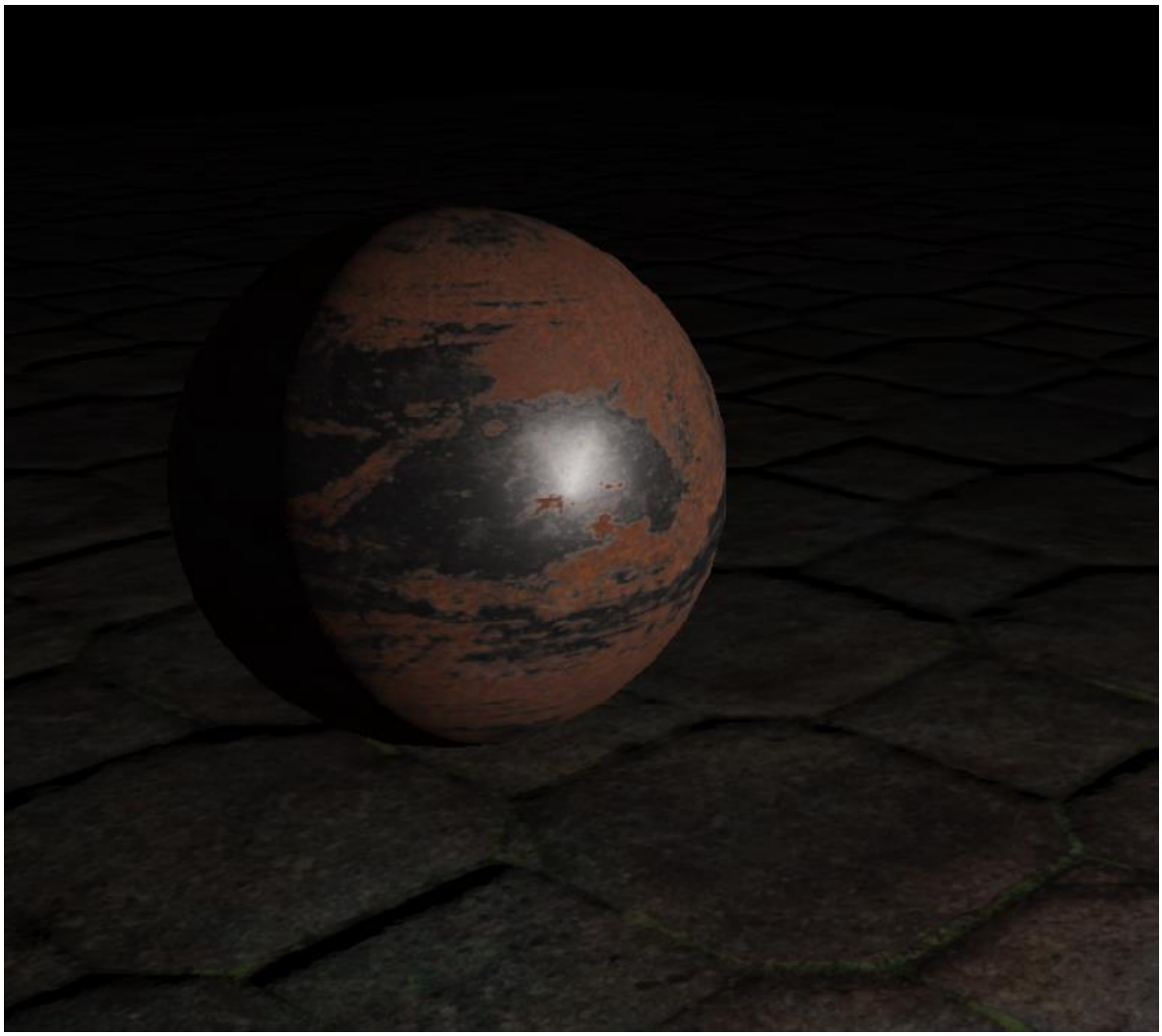
Kuviossa 14 on Cook-Torrancen mallin laskemiseen käytettyjä apumetodeja. DistributionGGX laskee normaalien hajautumisen pinnan normaalien,

puolivälivektorin ja karheuden avulla. GeometrySmith laskee GeometrySchlickGGX-metodin avulla geometriafunktion toteuttaen yhtälön (9).

```
float3 FresnelSchlick1(float HoV, float3 F0)
{
    return F0 + (1.0 - F0) * pow(1.0 - HoV, 5.0);
}
```

Kuvio 15. Fresnelin yhtälö HLSL-koodissa

Kuviossa 15 nähdään Fresnelin yhtälö Schlickin likiarvolla. Kuviossa 15 esitellyn metodin parametreinä on HoV, joka on pistetulo kappaleesta kameraan kulkevasta vektorista V, ja puolivälivektorista H. Parametri F0 on kappaleen metallisuus. Tässä metodissa toteutetaan yhtälö (11).



Kuvio 16. Cook-Torrancen toteutus ohjelmassa

Kuviossa 16 nähdään kuinka Cook-Torrancen BRDF-mallilla, joka on yhdistetty Lambertianin BRDF-mallin hajavaloon, saadaan aikaan hyvin realistisen näköinen lopputulos osittain sileästä metallista ja karheasta ruosteisesta metallista koostuvalla pinnalla.

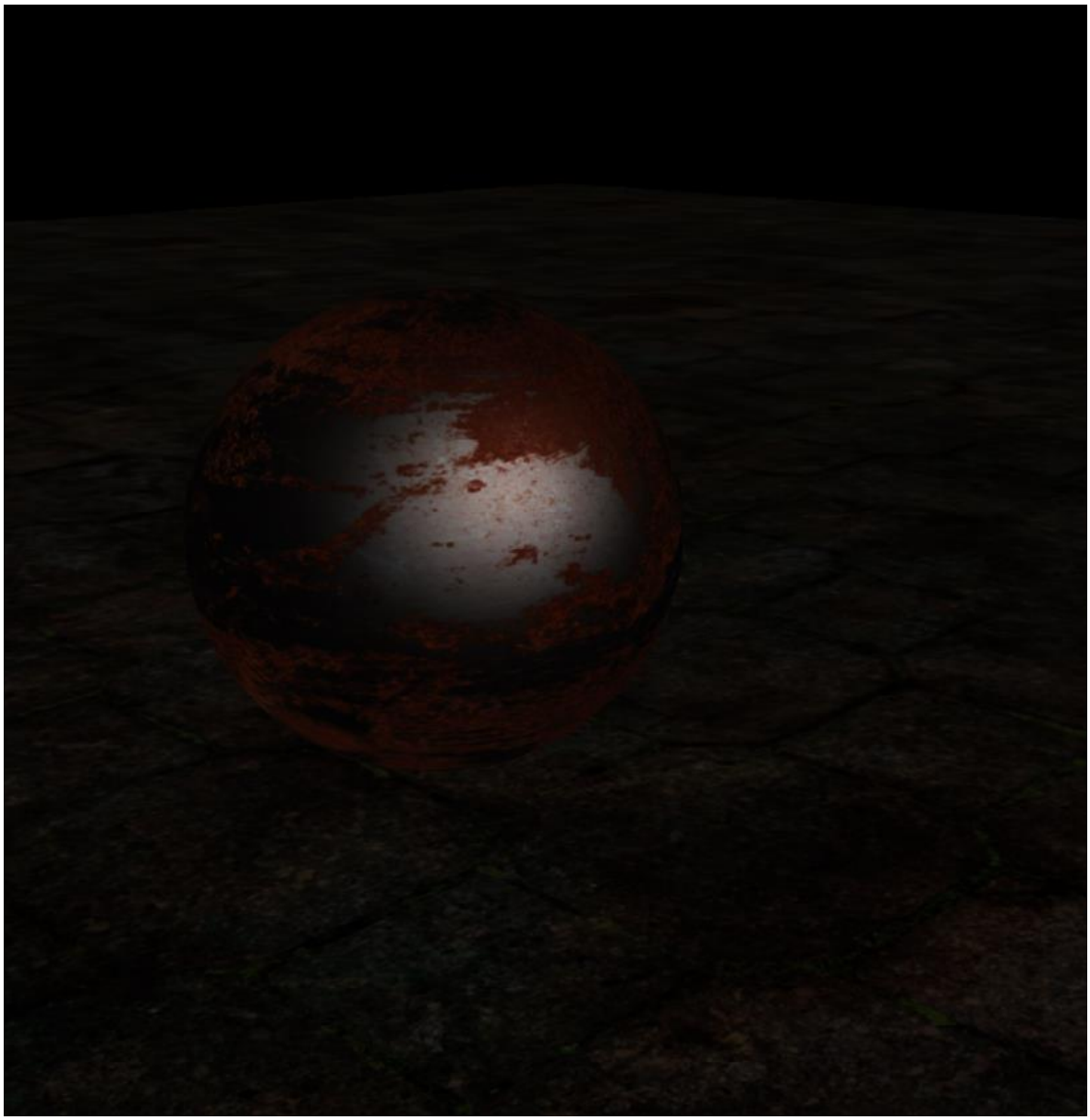
6.4 Wardin mallin toteutus

Wardin BRDF-malli on toteutettu yhtälön (13) mukaan varjostimessa.

```
float3 WardSpecular(float3 N, float3 V, float3 L, float a, float3 albedo, float metallic, float3 radiance, float3 tangent, float3 binormal)
{
    float3 ward = 0.0f.xxx;
    float _AlphaX = 1.0f;
    float _AlphaY = 1.0f;
    float3 F0 = 0.03f.xxx;

    float3 metallic1 = float3(metallic, metallic, metallic);

    F0 = lerp(F0, albedo, metallic1);

    float3 H = normalize(V + L);

    float NoL = dot(L, N);
    float HoN = dot(H, N);
    float HoV = dot(H, V);
    float VoN = dot(V, N);
    float AlphaX = dot(H, tangent) / _AlphaX;
    float AlphaY = dot(H, binormal) / _AlphaY;

    float3 F = FresnelSchlick1(max(HoV, 0.0), F0);

    float3 kS = F;
    float kD = float3(1.0f, 1.0f, 1.0f) - kS;
    kD *= 1.0 - metallic1;


    float3 spec =  F0 * sqrt(saturate(dot(NoL, VoN))) * exp(-2.0 * (AlphaX * AlphaX + AlphaY * AlphaY) / (1.0f + HoN));

    ward = kD * DiffuseLambertian(albedo) + spec * radiance * NoL;

    return ward;
}
```

Kuvio 17. Wardin mallin HLSL-koodi

Kuvion 17 esittelemässä Wardin BRDF-mallin metodissa käytetään kahta uutta parametriä. Ne ovat kappaleen pinnan bitangentit ja binormaalit. Bitangentti on pinnalla normaalista katsoen oikealle lähtevä vektori ja binormaali on ylöspäin lähtevä vektori. Näiden vektorien avulla saadaan käyttäjän määrittelemien _AlphaX- ja _AlphaY-nimisten muuttujien kanssa muutettua peiliheijastusta niin, että se vaikuttaa enemmän elliptimäiseltä.



Kuvio 18. Wardin BRDF-malli ohjelmassa

Kuviossa 18 Wardin BRDF-mallin toteutuksessa nähdään, että heijastuva valo on hieman ellipsimäinen, vaikka käyttäjän määrittelemät vakiot olivat molemmat 1. Käyttäjä pystyy halutessaan muuttamaan heijastuksen anisotrooppisuutta näitä vakioita muuttamalla.

6.5 Lambertianin mallin toteutus

Lambertianin mallia käytetään laskemaan vain hajavalon määrää. Sen mukaan valo heijastuu joka suuntaan yhtä paljon.

```
float3 DiffuseLambertian(float3 albedo)
{
    return albedo / PI;
}
```

Kuvio 19. Lambertianin mallin HLSL-koodi

Kuviossa 19 Lambertianin malli toteutetaan yksinkertaisesti niin, että pinnan väri jaetaan piillä. Toteutuksessa tähän lisätään vain ympäristövalo. Metodista nähdään yhtälön (17) toteutus.

Kuvio 20. Lambertianin BRDF-mallin toteutus ohjelmassa

Kuviossa 20 nähdään kuinka kappaleen väri pysyy tasaisena pallon pinnalla. Heijastuvaan valoon ei vaikuta katselukulma.

6.6 Oren-Nayarin mallin toteutus

Oren-Nayarin totetuksessa käytetään yhtälöä (18).

```
float3 Diffuse_OrenNayar(float3 DiffuseColor, float Roughness, float NoV, float NoL, float VoH)
{
    float a = Roughness * Roughness;
    float s = a; // / ( 1.29 + 0.5 * a );
    float s2 = s * s;
    float VoL = 2 * VoH * VoH - 1; // double angle identity
    float Cosri = VoL - NoV * NoL;
    float C1 = 1 - 0.5 * s2 / (s2 + 0.33);
    float C2 = 0.45 * s2 / (s2 + 0.09) * Cosri * (Cosri >= 0 ? rcp(max(NoL, NoV)) : 1);
    return DiffuseColor / PI * (C1 + C2) * (1 + Roughness * 0.5);
}
```

Kuvio 21. Oren-Nayarin mallin HLSL-koodi

Kuviossa 21 on metodi Oren-Nayarin BRDF-mallille, jota käytetään hajavalon esittämiseen. Metodin parametreinä on: pinnan väri, karheus, pistetulo vektoreista N ja V, pistetulo vektoreista N ja L, sekä pistetulo vektoreista V ja H.

Kuvio 22. Oren-Nayarin mallin toteutus ohjelmassa

Kuviossa 22 Oren-Nayarin mallin toteutuksesta voidaan huomata, että vaikka malli kuvaa kappaleesta heijastuvaa hajavaloa, niin heijastuvan valon määrä lisääntyy kappaleen pinnan kaareuden kasvaessa. Tämä on selvä ero Lambertianin malliin, jossa valon heijastuminen pysyy tasaisena kulmasta riippumatta.

7 Yhteenveto

Opinnäytetyössä saavutettiin alkuperäinen tavoite ja saatiin tehtyä toimiva grafiikkamoottori käyttämällä DirectX 11 -rajapintakokoelmaa. Grafiikkamoottoriin ei tullut huomattavasti eri ominaisuuksia, vaan se on tehty pelkästään tätä opinnäytetyötä varten.

Työssä päästään esittelemään kuutta eri BRDF-mallia. Vaikka malleja on todellisuudessa paljon enemmän, työssä tehdyt mallit ovat sellaisia, joita käytetään yleisimmin tietokonegrafiikassa. Työn laajuuden vuoksi kaikkia mahdollisia tietokonegrafiikassa käytettyjä BRDF-malleja ei päästy esittelemään.

Työn tuloksena voidaan päätellä, että eri BRDF-mallit sopivat eri käyttötarkoituksiin. Wardin mallista voidaan päätellä, että se sopii paremmin esimerkiksi hiusten tai kankaan heijastuksen laskemiseen, sillä näiden materiaalien heijastukset ovat yleensä elliptisiä. Cook-Torrancen toteutuksen perusteella havaitaan, että se on kaikista realistisimman näköinen. Se johtuen siitä, että malli on kehitetty seuraamaan fysiikan lakeja tarkasti. Blinn-Phongin ja Phongin BRDF-mallit ovat taas hyviä opetuskäyttöön, sillä niiden matematiikka on helposti ymmärrettävissä. Lopuksi hajavaloa kuvaavat Lambertianin ja Oren-Nayarin BRDF-mallit ovat molemmat hyviä vaihtoehtoja lähes kaikkiin käyttötarkoituksiin. Jos valaistuksen on tarkoitus seurata tarkemmin fysiikan lakeja, voidaan tällöin käyttää Oren-Nayarin mallia, mutta jos halutaan käyttää yksinkertaisempaa hajavaloa, tällöin on Lambertianin BRDF-malli sopiva siihen tarkoitukseen.

LÄHTEET

Assimp. Ei päiväystä. Open Asset Import Library. [Verkkosivu]. Assimp. [Viitattu 21.5.2017]. Saatavana: http://assimp.sourceforge.net/index.html

DirectX. Ei päiväystä. Microsoft DirectX. [Verkkosivu]. Computer Hope. [Viitattu 15.3.2017]. Saatavana: http://www.computerhope.com/jargon/d/directx.htm

Direct3D pipeline. Ei päiväystä. Graphics Pipeline. [Verkkosivu]. Microsoft. [Viitattu 1.4.2017]. Saatavana: https://msdn.microsoft.com/en-us/library/windows/desktop/ff476882(v=vs.85).aspx

Direct2D. Ei päiväystä. About Direct2D. [Verkkosivu]. Microsoft. [Viitattu 16.3.2017]. Saatavana: https://msdn.microsoft.com/en-us/library/windows/desktop/dd370987(v=vs.85).aspx

DXGI. Ei päiväystä. DXGI Overview. [Verkkosivu]. Microsoft. [Viitattu 18.3.2017]. Saatavana: https://msdn.microsoft.com/en-us/library/windows/desktop/bb205075(v=vs.85).aspx

Eimandar, P. 2013. DirectX 11.1 Game Programming. Packt Publishing Ltd. 24-25

Heiskanen, J. 2014. DirectX-pelimoottori. [Verkkojulkaisu]. Helsinki: Metropolia Ammattikorkeakoulu. Tietotekniikka. Opinnäytetyö. [Viitattu 20.4.2017]. Saatavana: http://www.theseus.fi/bitstream/handle/10024/75844/Heiskanen_Juha.pdf?sequence=1

Korhonen, S. 2013. Varjostinohjelmien käyttöön perustuvan reaaliaikaisen grafiikan ohjelmointi. [Verkko-julkaisu]. Jyväskylä: Jyväskylän yliopisto. Tietojenkäsittelytieteiden laitos, Tietojärjestelmätiede. Kandidaatin tutkielma. [Viitattu 3.4.2017]. Saatavana: https://jyx.jyu.fi/dspace/bitstream/handle/123456789/41048/Sauli%20Korhonen.pdf?sequence=1

Kurt, M. & Edwards, D. 4.6.2009. A Survey of BRDF Models for Computer Graphics. [Verkkojulkaisu]. Bornova: Ege University. International Computer Institute. [Viitattu 7.4.2017]. Saatavana: http://ube.ege.edu.tr/~kurt/Publications/SIGGRAPH_Computer_Graphics_2009_Kurt.pdf

Montes, R. Ureña, C. 2012. An Overview of BRDF Models. [Verkkojulkaisu]. Dept. Lenguajes y Sistemas Informáticos University of Granada, Spain. Saatavana: http://digibug.ugr.es/bitstream/10481/19751/1/rmontes_LSI-2012-001TR.pdf

Shader Models. Ei päiväystä. Shader Models vs Shader Profiles. [Verkkosivu]. Microsoft. [Viitattu 3.4.2017]. Saatavana: https://msdn.microsoft.com/en-us/library/windows/desktop/bb509626(v=vs.85).aspx

Shading2. 2016. More Shading and OpenGL. [Verkkosivu]. Opetussivusto. [Viitattu 30.4.2017]. Saatavana: http://jcsites.juniata.edu/faculty/rhodes/graphics/shading2.htm

Thibieroz, N. & Cebenoyan, C. 2010. DirectCompute Performance on DX11 Hardware. [Verkkojulkaisu]. Game Developers Conference. [Viitattu 5.4.2017]. Saatavana: http://developer.download.nvidia.com/presentations/2010/gdc/DirectCompute_Performance.pdf

Vries, J. 2017. PBR Lighting. [Verkkosivu]. PBR Lighting. [Viitattu 30.4.2017]. Saatavana: https://learnopengl.com/#!PBR/Lighting

Wynn, C. Ei päiväystä. An Introduction to BRDF-Based Lighting. [Verkko-julkaisu]. NVIDIA Corporation. [Viitattu 15.4.2017]. Saatavana: https://pdfs.semanticscholar.org/023a/5c184b8cb9e9c5f2ac61fa679fedc1a478f8.pdf

XAudio2. Ei päiväystä. XAudio Introduction. [Verkkosivu]. Microsoft. [Viitattu 15.3.2017]. Saatavana: https://msdn.microsoft.com/en-us/library/windows/desktop/ee415813(v=vs.85).aspx

XInput. Ei päiväystä. XInput and DirectInput. [Verkkosivu]. Microsoft. [Viitattu 16.3.2017]. Saatavana: https://msdn.microsoft.com/en-us/library/windows/desktop/ee417014(v=vs.85).aspx

Zink, J., Pettineo, M. & Hoxley, J. 2016. Practical Rendering and Computation with Direct3D 11. 1-2. CRC Press.